Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Kateřina Topilová

Překladová paměť s alignmentem vět

Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Jiří Semecký, Ph.D.
Studijní program: Informatika, Programování

2008

Poděkování

Děkuji panu Mgr. Jiřímu Semeckému, Ph.D. za vedení této práce a spoustu trpělivosti, kterou při tom prokázal.
Dále bych chtěla poděkovat svým rodičům, bez jejichž podpory by tato práce nikdy nevznikla, a v neposlední řadě děkuji všem, kdo práci přečetli a podíleli se na odstranění chyb.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 12. 12. 2008 Kateřina Topilová

Obsah

Název práce: Překladová paměť s alignmentem vět
Autor: Kateřina Topilová
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Mgr. Jiří Semecký, Ph.D.
e-mail vedoucího: semecky@ufal.mff.cuni.cz
Abstrakt: Překladová paměť (TM) je soubor záznamů, kde je zdrojový segment asociován se svým překladem v jednom nebo i více jazycích. Systém s překladovou pamětí v této databázi vyhledává podobnosti mezi zadaným segmentem a zdrojovou částí již uložených záznamů. Některé systémy vyhledávají pouze přesné shody segmentů, jiné používají různé „fuzzy" algoritmy a vyhledávají segmenty do určité míry podobné. Alignment popisuje vztah, který je mezi textem a jeho překladem. Říká nám, který segment zdrojového textu odpovídá kterému segmentu cílového textu. Předmětem této práce je program, který poskytne prostředky pro práci s překladovou pamětí a umožní vytvoření nové paměti z už přeložených dokumentů pomocí alignmentu.
Klíčová slova: překladová paměť, alignment, fuzzy algoritmy

Title: Translation Memory with Sentence Alignment
Author: Kateřina Topilová
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Jiří Semecký, Ph.D.
Supervisor's e-mail address: semecky@ufal.mff.cuni.cz
Abstract: Translation memory (TM) is a set of units, in which the source segment is associated with its corresponding translation in one or more target languages. In this database the translation memory system searches for similarities between a given segment and the source part of already saved units. Some systems search only for exact matches between segments, other use various „fuzzy" algorithms allowing them to search even for similar matches. Alignment describes the relation between the source text and its translation. It says which segment of the source text corresponds to which segment of the target text. The object of this work is to design and implement a program, that will provide its users with the means to work with a translation memory and will allow them to create a new memory from already translated documents by the means of alignment.
Key words: translation memory, alignment, fuzzy search

Kapitola 1
Úvod

Jedním z nejlepších a nejdůležitějších zdrojů informací, ke kterým může mít překladatel při tvorbě překladu přístup, je překladová paměť s velkým množstvím uložených předchozích překladů. Může mu poskytnout obvyklou firemní terminologii a styl, který nenalezne v žádném slovníku, ušetřit čas a zajistit konzistenci překladu v případě, že na něm pracuje více osob. Toto je však možné pouze, pokud je na již přeložených textech proveden alignment a je tak umožněno vyhledání odpovídajícího segmentu ve zdrojovém i cílovém textu.

Na rozdíl od systémů plně automatického překladu jsou nástroje s překladovou pamětí v překladatelství a lokalizaci poměrně hojně využívány. Patří do skupiny, tzv. CAT (počítačem asistovaný překlad) tools, což je dosti široký pojem, který zahrnuje například i kontroly pravopisu a gramatiky, různé elektronické slovníky a databáze termínů, full-textové vyhledávače apod. Nástroje s překladovou pamětí slibují zrychlit a zlevnit překlad tím, že si zapamatují každou větu, kterou překladatel přeloží, a když na stejnou nebo podobnou větu narazí v budoucnu, nabídnou její uložený překlad. Toto je velmi výhodné pro velké společnosti, které často vytvářejí nové verze svých produktů. V dokumentacích a jiných materiálech se totiž často pro novou verzi objeví jen málo nových vět nebo odstavců v porovnání s odstavci již dříve přeloženými. Překladové paměti těchto materiálů pak mají hodnotu v řádech desítek až stovek tisíc. Není proto divu, že na správu a tvorbu těchto pamětí vzniklo v poslední době mnoho programů, z nichž ty nejlepší jsou bohužel komerční a dosti nákladné. Předmětem této práce je program, který poskytne prostředky pro práci s překladovou pamětí, umožní vytvoření nové paměti z už přeložených dokumentů pomocí alignmentu, a bude bez poplatků a omezení používání.

1.1 Přehled dalších kapitol

Ve druhé kapitole jsou popsány cíle práce a především funkční požadavky, ze kterých jsem vycházela při návrhu a implementaci.

Analýza řešení ve třetí kapitole uvádí existující programy a jejich stručný popis, přibližuje standardizovaný formát TMX a vysvětluje algoritmy použité při implementaci.

Čtvrtá kapitola se věnuje návrhu aplikace s ohledem na funkční požadavky, popisuje její rozdělení na jádro a moduly a blíže se věnuje jejich implementaci. Je v ní zmíněno také rozdělení zdrojového kódu do souborů a kompilace.

Poslední kapitola hodnotí výsledek práce po jejím dokončení, srovnává s již dříve zmíněnými aplikacemi a uvádí možnosti dalšího vývoje.

Úplný závěr práce patří přehledu použitých zdrojů a přílohám – ukázce dokumentu ve formátu TMX a popisu obsahu přiloženého CD.

Kapitola 2
Cíle práce

2.1 Motivace

Přestože používání systému s překladovou pamětí má mnoho výhod, jsou i překladatelé, kterým se pravidelné používání takového nástroje nevyplatí. Jejich zákazníci použití takového systému vyžadují jen výjimečně, jejich překlady jsou různorodé, nebo i umělecké. Málo často opakují stejné věty, a pokud by používali ke všem svým překladům systém s překladovou pamětí, narostla by do velkých objemů jen s velmi malým užitkem. Ten by navíc byl ještě snížen časem, který by překladatel musel věnovat přidávání každé věty, a časem, který by zabralo prohledávání takové paměti. Ale i takový překladatel občas dostane zakázku, kdy je po něm použití překladové paměti vyžadováno. Zpravidla je nějaká dodána pro přidání zadaných vět, nebo může být i požadavek na její vytvoření. Pokud nechce zakázku odmítnout, nezbude mu nic jiného než nalézt systém, který mu toto umožní, nebude náročný na použití a bude pokud možno zadarmo. Protože jsem nenašla systém, který by mým požadavkům vyhovoval, rozhodla jsem se ho vytvořit.

2.2 Překladová paměť

Překladová paměť (TM), nebo také překladová databáze, je soubor záznamů, kde je zdrojový segment asociován se svým překladem v jednom, nebo i více jazycích. Systém s překladovou pamětí v této databázi vyhledává podobnosti mezi zadaným segmentem a zdrojovou částí již uložených záznamů. Některé systémy vyhledávají pouze přesné shody segmentů, jiné používají různé „fuzzy“ algoritmy a vyhledávají segmenty do určité míry podobné.

Segment je část textu, většinou věta, ale mohou to být i jednotlivá slova nebo víceslovné fráze. V některých případech může být segmentem i celý odstavec.

Záznamu překladové paměti se říká *překladová jednotka* (TU). Obsahuje segment zdrojového textu a jeden nebo více segmentů textu v cílovém jazyce nebo jazycích. Překladová jednotka dále obvykle obsahuje údaje o textu, který obsahuje. Zejména je

to typ segmentu a druh textu, ze kterého pochází. Dále může obsahovat informace o tom, kdy byla vytvořena, či kdo ji vytvořil nebo údaj o tom, kolikrát již byla použita, a další.

Přesná shoda (exact, perfect match) při vyhledávání nastává, pokud zdrojový segment nalezené překladové jednotky přesně odpovídá segmentu, který chceme přeložit.

„Fuzzy" shoda (fuzzy match) nastává, pokud se tyto dva segmenty neshodují úplně, ale jsou si pouze podobné. Překladatel pak může segment pouze upravit, nemusí ho psát znovu.

Některé překladové paměti používají tzv. „*markup*" k zaznamenání a uchování původního formátu dokumentu. Po průchodu překladovou pamětí má pak dokument v cílovém jazyce zachováno původní formátování. Uchovávání „markup" kódů ale může mít za následek snížení efektivity vyhledávání a snížení frekvence výskytu přesných shod. Pokud se totiž dva segmenty liší formátováním jsou obvykle považovány za „fuzzy" shodu.

Překladové paměti se vyskytují v různých formátech, z nichž nejčastější jsou formáty *XLIFF* (XML Localization Interchange File Format) [1], který se používá zejména při procesu lokalizace, a *TMX* (Translation Memory eXchange format) [2], který poskytuje standardizované prostředky k popisu překladové paměti, jež je používána různými systémy, bez ztráty důležitých dat.

2.3 Alignment vět

Alignment popisuje vztah, jenž je mezi textem a jeho překladem. Říká nám, který segment zdrojového textu odpovídá kterému segmentu cílového textu. Alignment je možné (a smysluplné) provádět na dokumentech, odstavcích, větách a slovech.

Aby bylo možné provádět alignment na větách, je na textu třeba nejprve provést *segmentaci*, tj. oddělit od sebe slova, rozpoznat začátky a konce vět a odstavců, případně i další entity jako jsou jména, data, čísla.

Někdy může být užitečné vědět, jak byl dokument obsažený v překladové paměti segmentován. K ukládání a sdílení pravidel segmentace slouží formát *SRX* (Segmentation Rules eXchange format) [3].

2.4 Funkční požadavky

Požadavky na funkčnost aplikace jsem se pro větší přehlednost rozhodla zapsat do bodů. Program by měl umět následující:

- operace s překladovou pamětí
  - otevření překladové paměti ve formátu TMX
  - prohlížení překladové paměti
  - přidání jednotky do paměti, úprava jednotky, smazání jednotky
  - sloučení dvou překladových pamětí
  - export cílového textu
- operace s překladovým projektem
  - založení nového projektu – zahrnuje otevření zdrojového souboru, jeho rozložení na segmenty k překladu a otevření nebo vytvoření překladové paměti
  - uložení překladového projektu
  - překlad pomocí paměti – automatické vyhledávání v paměti, fuzzy a perfect match, update paměti
  - automatický překlad – pokus o automatické vyhledání všech segmentů v paměti a jejich následný export do formátu zdrojového souboru
- operace s alignment projektem
  - založení nového projektu – otevření zdrojového a cílového souboru, jejich rozložení na segmenty a následný alignment, vytvoření překladové paměti
  - úprava alignmentu – sloučení dvou překladových jednotek nebo pouze jejich zdrojových nebo cílových částí
  - uložení do formátu TMX
- další požadavky
  - spustitelné pod operačním systémem Windows i Linux
  - nevázané na další aplikace
  - modularita a snadná rozšiřitelnost

Kapitola 3
Analýza řešení

3.1 Přehled existujících programů

Systémů s překladovou pamětí existuje poměrně velké množství, ale většina z nich je komerčních a pro příležitostné použití se investice do nich nevyplatí. Z nejznámějších bych jmenovala například SDL Trados, Déjà Vu, MultiTrans, nebo Wordfast. Tato velká softwarová řešení obsahují takové množství dalších funkcí a nástrojů, že je v podstatě nelze srovnávat s volně šiřitelnými aplikacemi, proto se dále budu věnovat pouze programům, které jsou dostupné zdarma.

Anaphraseus[4]

Anaphraseus je skupina maker, dostupná jako rozšíření kancelářského balíku OpenOffice.org. Původně pracoval s překladovou pamětí Wordfastu, ale nyní již umí i importovat a exportovat soubory TMX. Nepodporuje alignment.

OmegaT+[5]

OmegaT+ je projekt psaný v jazyce Java, který se snaží spojit několik programů asistovaného překladu. K jeho efektivnímu fungování je ale třeba použít další programy, které zajišťují například úpravu segmentace nebo alignment.

Open Language Tools[6]

Open Language Tools je sada programů, které mají za cíl ulehčit lokalizaci. Jejich cílovou skupinou jsou tedy spíše lidé, kteří překládají uživatelská rozhraní a dokumentace aplikací. Tomu odpovídá také to, že jádro programu je založeno na XLIFF editoru. V současné době podporují i import/export TMX souborů a alignment, který je prováděn zvláštním programem, který je ale také součástí balíku. Open Language Tools jsou psané v Javě.

Transolution[7]

Transolution je systém asistovaného překladu napsaný v Pythonu. Jeho jádrem je také XLIFF editor, podporuje funkce překladové paměti a je velmi modulární. Zatím neobsahuje filtry, které by do něho přidaly podporu TMX, ani neprovádí alignment.

Translate Toolkit & Virtaal[8]

Translate Tookit je lokalizační nástroj, který obsahuje také API pro vytváření dalších programů a nástrojů, Virtaal je pak jedním z těchto nástrojů. Pracuje se soubory ve formátu PO, ale obsahuje různé konvertory z i do dalších formátů, kromě jiných také TMX a XLIFF. Mezi jeho funkcemi není alignment. Translate Toolkit je také psaný v Pythonu.

Transit[9]

Transit je systém s překladovou pamětí pro Windows. Má své vlastní formáty, a přestože uvádí kompatibilitu s formátem TMX, nepodařilo se mi ji ověřit. Neumožňuje alignment již dříve přeložených souborů.

3.2 Formát TMX

3.2.1 Co je TMX a proč ho používat

Formát TMX byl vyvinut v roce 1998 komisí pro standardizaci OSCAR asociace LISA (Localization Industry Standards Association) [10] jako reakce na stále častější používání systémů s překladovou pamětí. Je to otevřený XML standard, který umožňuje jednodušší výměnu překladových pamětí mezi jednotlivými programy i překladateli a zaručuje, že při tomto procesu nedojde k žádné, nebo jen velmi malé, ztrátě kritických dat. Mezi výhody jeho použití patří:

- flexibilita – není nutné používat pouze jeden nástroj nebo jednoho překladatele/agenturu, můžete si vybrat to nejlepší pro konkrétní úkol;
- větší výběr – už nemusíte využívat služby pouze těch překladatelů/agentur, které používají jeden konkrétní nástroj;
- větší kontrola nad pamětí – vytvoření a rozšíření překladové paměti je dosti drahý proces a překladová paměť je tak jedním z aktiv společnosti. Nyní není nutné přenechat celý její obsah třetí straně bez možnosti kontroly.

Zatím poslední verzí je TMX 1.4b [11].

3.2.2 Struktura

TMX je XML dokument a používá také různé ISO standardy, například pro čas, datum, jazykové kódy apod. Jména elementů a atributů se píší malými písmeny.

TMX je rozdělené do dvou částí: v první jsou elementy, které poskytují informace o paměti a dokumentu jako celku a o překladových jednotkách, tzv. strukturální elementy, ve druhé pak nízkoúrovňové informace o formátování obsahu segmentu, tzv. vložené elementy.

Strukturální elementy tvoří základní kostru dokumentu. Celý je uzavřen do kořenového elementu <tmx>, který dále obsahuje elementy <header> a <body>.

Element <header> obsahuje informace o celém dokumentu – např. jakým nástrojem byl vytvořen, kdo a kdy ho vytvořil, typ segmentů, které obsahuje, zdrojový jazyk a další. Dále může obsahovat elementy <note>, které slouží jako komentáře, <prop>, které definují různé vlastnosti dokumentu, které ale nejsou definované formátem, a <ude>, které mohou obsahovat uživatelem definované znaky a jejich mapování z Unicode do uživatelem definovaného kódování.

Element <body> obklopuje hlavní data – soubor překladových jednotek v elementech <tu>. Tyto jednotky nejsou v žádném speciálním pořadí, tzn. nejsou nijak tříděné, ani nemusí být ve stejném pořadí, ve kterém se vyskytly v originálním dokumentu.

Element <tu> obsahuje data překladové jednotky. Jsou to nejen informace o jejím vytvoření, ale také komentáře a další její vlastnosti v elementech <note> a <prop>. Samotný obsah překladové jednotky je uzavřen v elementech <tuv>. Překladová jednotka je úplná, pokud obsahuje alespoň dva elementy <tuv>, jeden ve zdrojovém jazyce a druhý v cílovém jazyce.

Element <tuv> značí variantu překladové jednotky a stejně jako celá překladová jednotka obsahuje informace o svém vytvoření a další informace v elementech <note> a <prop>. Navíc obsahuje povinný údaj o tom v jakém jazyce je psána. Element <tuv> také konečně obsahuje segment v udaném jazyce, což je text uzavřený v elementu <seg>.

Vložené elementy se mohou vyskytovat pouze uvnitř elementu <seg> a označují, kde se v textu objevily formátovací a kontrolní značky. Tyto značky mohou být tří typů a TMX poskytuje prostředky k označení každého z nich:

- Značky, které začínají nebo končí instrukci, a jejich začátek i konec se vyskytují ve stejném segmentu, jsou ohraničeny elementy <bpt> a <ept>.

Př.: <seg>
The
<bpt i="1" x="1"></bpt>
big
<ept i="1"></ept>
black cat.
</seg>

- Značky, které začínají nebo končí instrukci, ale jejichž druhá část se nevyskytuje ve stejném segmentu, například kvůli segmentaci, jsou ohraničeny elementem <it>.

Př.: <seg>
<it pos="begin" x="1"></it>
First segment.
</seg>
…
<seg>
Second segment.
<it pos="end" x="1"></it>
</seg>

- Značky, které znamenají samostatně stojící instrukci a nepotřebují ukončení, nebo nerozpoznané značky jsou ohraničeny elementem <ph>.

Př.: <seg>
The icon
<ph x="1"></ph>
represents a conditional node.
</seg>

K vloženým elementů ještě dále patří dva speciální elementy: <hi> a <sub>.

Element <hi> ohraničuje část textu, která má speciální význam, například jméno které se nemá překládat, nebo terminologii, kterou si přejeme ověřit.

Př.: <seg>
<hi type="name" x="1">George Bush</hi>
visited Europe.
</seg>

Element <sub> se používá k oddělení textu vnořeného do sekvence formátovacích značek.

Př.: <seg>
See the
<bpt i="1" type="link"></bpt>
Notes
<ept i="1"></ept>
for more details.
</seg>

Použití elementu <sub> však může způsobit problémy při převodu do jiného nástroje, protože některé nástroje vyjímají vnořený text jako další segment.
V příloze B naleznete ukázku dokumentu ve formátu TMX.

3.3 Použité algoritmy

3.3.1 Fuzzy vyhledávání a Levenshteinova vzdálenost

Fuzzy vyhledávání, nebo také přibližné vyhledávání, se používá při vyhledávání řetězců, které jsou podobné zadanému řetězci. Mezi jeho aplikace patří zejména kontrola pravopisu, dobývání znalostí, vyhledávání podobných řetězců nukleových kyselin v komputační biologii (například po mutaci), a také právě vyhledávání podobného segmentu v překladové paměti.
Základním prvkem přibližného vyhledávání je podobnostní funkce, která páru řetězců přiřadí hodnotu, které udává, jak moc se dané dva řetězce liší. Přiřazená hodnota se samozřejmě liší podle použité podobnostní funkce. Mezi nejznámější patří Hammingova vzdálenost na řetězcích stejné délky, která vzdálenost mezi těmito řetězci udává jako minimální počet substitucí, které musíme provést, abychom z jednoho řetězce dostali druhý. Dále Levenshtainova vzdálenost, která počítá vzdálenost jako minimální počet operací, které musíme provést, abychom jeden řetězec převedli na druhý, ale za operaci už považuje nejen substituci jednoho znaku, ale také smazání jednoho znaku a vložení jednoho znaku. Řetězce tedy nemusí být stejné délky. Rozšířením Levenshtainovy vzdálenosti je Damerau-Levenshtainova vzdálenost, která za operaci považuje i prohození dvou sousedních znaků. Dále se

ještě běžně používá tzv. n-gramová vzdálenost, která převádí řetězce na n-gramy a vytváří jejich vektorovou reprezentaci. Na základě podobnosti mezi těmito reprezentacemi pak rozhoduje, zda si jsou řetězce podobné či nikoli.

Levenshtainova vzdálenost a různé její modifikace je však nejrozšířenější a je považována za standardní.

Samotný algoritmus je založen na technice dynamického programování, kde je funkce *f (i, j)* iterativně spočítána pomocí následujících rekurentních vztahů[12]:

$$f(0, 0) = 0$$

$$f(i, j) = \min [f(i-1, j) + 1,$$
$$f(i, j-1) + 1,$$
$$f(i-1, j-1) + d(s_i, t_j)]$$

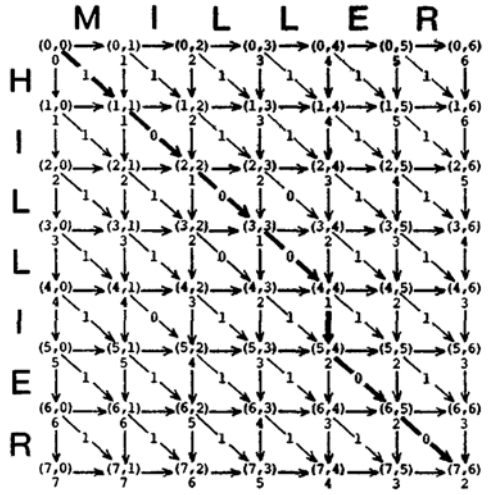
kde

$f(i, j)$ je vzdálenost pro nejlepší shodu řetězců $s_1s_2s_3\ldots s_i$ a $t_1t_2t_3\ldots t_j$

$d(s_i, t_j) = 0$, když $s_i = t_j$

$= 1$ jinak

Tato metoda může být také reprezentována jako hledání nejkratší cesty v grafu:



Obr. 1: Příklad porovnání dvou řetězců. Každý uzel je označen svým příslušným *(i, j)* a pod ním je hodnota *f (i, j)*. Hodnoty na diagonálách jsou *d* (s_i t_j). Podél vertikálních a horizontálních hran je penalizace, která je v tomto případě 1. Zde je minimální vzdálenost mezi řetězci 2 a může být zjištěna jako nejkratší cesta v tomto grafu (označená silně).
Převzato z Hall&Dowling(1980)[12]

3.3.2 Statistický alignment

Problém automatického alignmentu spočívá v tom, že jedna věta zdrojového textu nemusí odpovídat jedné větě textu cílového. Překlady, kdy jedna věta odpovídá dvěma, či dvě věty odpovídají jedné jsou dosti běžné, zejména u kvalitního překladu,

kde se dává přednost překladu významu před doslovným překladem. Méně často se mohou dokonce vyskytnout případy, že dvě věty odpovídají dvěma větám, ale nelze je od sebe oddělit, či případy, kdy segment v jenom jazyce odpovídá třem, nebo i více segmentům v jazyce druhém. Přesto není překvapivé (a bylo i experimentálně prokázáno – Brown, Lai, Mercer(1991) [13]), že délka zdrojové a cílové věty spolu určitým způsobem souvisí a většina algoritmů toho také využívá.

Automatický alignment pro účely překladové paměti se poněkud liší od alignmentu pro automatický překlad založený na statistice. Do překladové paměti chceme většinou zařadit pouze segmenty, u kterých jsme si jisti, že si 100% odpovídají, a proto bývá kontrolován člověkem. Pro účely automatického překladu, či jiné činnosti založené především na statistice je však taková kontrola velmi neefektivní z důvodu velkého rozsahu textů. Rozsah a kvalita textů se také velmi liší. Pro účely statistiky jsou tyto texty dlouhé (i několik milionů vět) a jejich kvalita je zpravidla nižší, protože nejsou kontrolovány (mohou dokonce samy být již výsledkem automatického překladu), případně jsou získávány pomocí OCR a jejich čištění se nevyplatí. Na druhou stranu texty pro překladovou paměť jsou zpravidla kratší – typicky maximálně v řádu několika desítek stran – a „čisté“ (kromě formátování), protože překladatel dnes dostává zdrojové texty v elektronické podobě.

K automatickému alignmentu existuje několik přístupů. V našem případě předem nevíme, v jakých jazycích překladová paměť bude, proto byly předem vyloučeny všechny metody, které se spoléhají na znalost jazykových jevů a/nebo na slovník. Dále nebyly zvažovány metody, které byly vyvinuty s ohledem na „šum“ v textech (vzniklý například OCR), protože jak jsem dříve již uvedla, texty převáděné do pamětí nejsou většinou tímto způsobem získávány.

Brown, Lai a Mercer (1991) používají metodu založenou na skrytých Markovových modelech pro generování alignovaných párů. Délky vět počítají na slova. Představují si, že

> *„…věty byly vygenerovány dvojicí náhodných jevů, z nichž první vytvořil sekvenci alignmentů a druhý vybral délky jednotlivých vět v každém z nich.“* [13]

Kay a Röscheisen (1993) také svou metodu zakládají na slovech. Využívají částečného alignmentu na slovech ke zlepšení alignmentu na úrovni vět. Ten dále

využívají ke zlepšení již získaného alignmentu slov a takto dále iterují. Alignment na slovech je získán tak, že

> *„…předpokládáme množinu párů slov, o kterých se domníváme, že si navzájem odpovídají, podle podobností mezi jejich rozděleními v obou textech.“* [14]

Gale a Church (1991) popisují metodu založenou na jednoduchém statistickém modelu délky znaků:

> *„Program využívá faktu, že delší věty v jednom jazyce bývají překládány jako delší věty v jazyce druhém, a že věty kratší bývají překládány jako kratší. Každé navrhované shodě vět je přiřazeno pravděpodobnostní skóre založené na vážené odchylce délek těchto dvou vět (ve znacích) a rozptylu této odchylky. Toto pravděpodobnostní skóre je poté použito v rámci dynamického programování k nalezení nejvíce pravděpodobného alignmentu vět.“*[15]

Po uvážení jsem si vybrala metodu, kterou popisují Gale a Church, protože mne nejvíce přesvědčila publikovanými výsledky. Je rychlejší než metoda, kterou navrhuje Kay a Röscheisen, protože neprovádí několik iterací nad daty, a Brown, Lai, Mercer udávají pouze úspěšnost na alignmentu 1-1 a nezmiňují se o složitějších případech. Nevýhodou této metody je snad pouze to, že vyžaduje, aby oba texty měly stejný počet odstavců. Pro texty, které chceme převést do překladové paměti, toto ale nepředstavuje problém. Dále se tedy budu věnovat pouze popisu této metody.

Nejprve se provádí alignment na odstavcích, ve druhém kroku pak alignment na větách uvnitř těchto odstavců. Problémem je rozpoznávání vět (tečka nemusí znamenat konec věty, ale také datum, zkratku, nebo číslo), ale protože je v překladové paměti alignment ještě většinou ručně kontrolován, mohou být chyby plynoucí ze špatného rozpoznání konce věty opraveny a nebudeme tedy s nimi dále počítat.

Obecně je zde použita míra vzdálenosti, která porovnává dva jednotlivé elementy v posloupnostech, a algoritmus dynamického programování k minimalizaci celkových vzdáleností mezi alignovanými elementy. Míra vzdálenosti je založena na předpokladu, že každý znak v jednom jazyce L_1, způsobuje existenci náhodného

počtu znaků v druhém jazyce L_2 a je odhadnuta na *–log Pr(shoda | δ)*, kde δ závisí na l_1 a l_2, což jsou délky částí textu, které zvažujeme. Předpokládá se, že tyto náhodné veličiny jsou nezávislé a mají stejné normální rozdělení. Model je pak specifikován střední hodnotou c, což je očekávaný počet znaků v L_2 na znak v L_1, a rozptylem počtu znaků v L_2 na znak v L_1, s^2. δ je pak definována jako $(l_2 - l_1 c)/\sqrt{l_1 s^2}$, takže má normální rozdělení se střední hodnotou 0 a odchylkou 1. Gale a Church určují parametry c a s^2 empiricky jako $c \approx 1$ a $s^2 \approx 6.8$ a říkají, že jsou použitelné pro většinu párů evropských jazyků. Protože však u překladové paměti neznáme jazykové páry, jdou v programu parametry c a s^2 zadat tak, aby byly použitelné například i pro čínštinu. Dále podle Bayesova teorému:

$$Pr(shoda \mid \delta) = Pr(\delta \mid shoda) \cdot Pr(shoda)$$

Podmíněná pravděpodobnost $Pr(\delta \mid shoda)$ může být odhadnuta jako:

$$Pr(\delta \mid shoda) = 2(1 - Pr(|\delta|))$$

kde $Pr(|\delta|)$ je pravděpodobnost, že náhodná veličina z se standardním normálním rozdělením má velikost alespoň jako |δ|.

$$Pr(\delta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\delta} e^{-z^2/2} dz$$

Hodnoty *Pr(shoda)* byly určeny jako:

Alignment	*Pr(shoda)*
1-1	0,89
1-0 nebo 0-1	0,0099
1-2 nebo 2-1	0,089
2-2	0,011

Tab. 1: Empirické určení *Pr(shoda)*

Vzdálenostní funkce d je definována tak, aby dovolovala vkládání, mazání, shodu a další operace:

d(x_1, y_1; 0, 0) je cena za shodu x_1 a y_1

d(x_1, 0; 0, 0) je cena za smazání x_1

$d(0, y_1; 0, 0)$ je cena za vložení y_1

$d(x_1, y_1; x_2, 0)$ je cena za sloučení x_1 a x_2 do y_1

$d(x_1, y_1, 0, y_2)$ je cena za rozšíření x_1 na y_1 a y_2

$d(x_1, y_1; x_2, y_2)$ je cena za sloučení x_1 a x_2 a shodu s y_1 a y_2.

Mějme $s_1, s_2, s_3, \ldots, s_i$ věty v jednom jazyce a $t_1, t_2, t_3, \ldots, t_j$ jejich překlady, $D(i, j)$ minimální vzdálenost mezi větami $s_1, \ldots, s_i$ a překlady $t_1, \ldots, t_j$ podle maximálně pravděpodobného alignmentu. $D(i, j)$ se spočítá minimalizací přes šest výše uvedených případů a je definována následující rekurentní formulí s počáteční podmínkou $D(i, j) = 0$.

$$D(i, j) = min \begin{cases} D(i, j-1) + d(0, t_j; 0, 0) \\ D(i-1, j) + d(s_i, 0; 0, 0) \\ D(i-1, j-1) + d(s_i, t_j; 0, 0) \\ D(i-1, j-2) + d(s_i, t_j; 0, t_{j-1}) \\ D(i-2, j-1) + d(s_i, t_j; s_{i-1}, 0) \\ D(i-2, j-2) + d(s_i, t_j; s_{i-1}, t_{j-1}) \end{cases}$$

Jednoduchou úpravou se dá zbavit požadavku na stejný počet odstavců, ale tato úprava má za následek podstatné snížení přesnosti.

Kapitola 4

Návrh a implementace

4.1 Rozbor a návrh

Jedním z požadavků na aplikaci byla její spustitelnost na operačních systémech Windows i Linux. Bylo tudíž třeba vybrat přenositelné knihovny, které by umožňovaly následující:

- podpora Unicode;
- přenositelné načítání modulů;
- tvorbu přenositelného grafického uživatelské rozhraní.

Vybírala jsem mezi knihovnami GTK+[16] a wxWidgets[17]. Nakonec jsem se rozhodla pro GTK+ s aplikačním rozhraním Gtkmm[18], protože mi více vyhovoval její způsob práce s Unicode.

Celá aplikace je napsána v jazyce C++ s použitím standardních knihoven STL a již zmíněných GTK+. Kód je zabalen do tříd a každá entita má svou odpovídající třídu. Aplikace je rozdělená na následují logické celky:

- jádro, do kterého patří reprezentace projektů, překladové paměti, nastavení a rozhraní pro moduly;
- moduly;
- uživatelské rozhraní.

V následujících kapitolách se budu věnovat pouze stručnému popisu tříd a vztahů mezi nimi. Podrobnější popis je možné nalézt v programátorské dokumentaci, která je součástí této práce a nachází se na přiloženém CD.

4.2 Jádro aplikace

Jádro aplikace je tvořeno čtyřmi základními třídami, které představují projekty, překladové paměti, nastavení a rozhraní pro připojené moduly. Protože moduly používají některé třídy a funkce z jiných částí aplikace, je celé jádro sdílená knihovna, ke které má přístup uživatelské rozhraní a právě moduly.

4.2.1 Projekty (třída CProject)

Třída `CProject` je z hlediska logického návrhu abstraktní a poskytuje základní rozhraní pro práci se všemi projekty, které jsou od ní odvozené. Všichni potomci této třídy musí definovat metody, které umožňují jejich založení (`NewProject`), otevření (`OpenProject`) a export paměti projektu (`ExportMemory`). Třída dále obsahuje metody, které umožňují práci s projektem, jako například uložení projektu (`SaveProject` a `SaveProjectAs`), převod projektu na překladovou paměť (`ConvertProjectToMemory`), uložení přidružené paměti (`SaveTM` a `SaveTMAs`), sloučení překladové paměti projektu s jinou pamětí (`MergeTMs`), otevření a přidružení překladové paměti (`OpenTM`), základní práci s překladovými jednotkami projektu (`DeleteUnit`, `AddUnit`, `ChangeSource`, `ChangeTarget`, `MergeUnits`, `BreakUnit`) a změnu nastavení (`ChangeSettings`). Členy třídy jsou ukazatelé na data projektu, paměť, nastavení a rozhraní pro moduly.

Od třídy `CProject` jsou odvozeny třídy `CTranslationProject`, která reprezentuje překladový projekt a `CAlignmentProject`, která představuje alignment projekt.

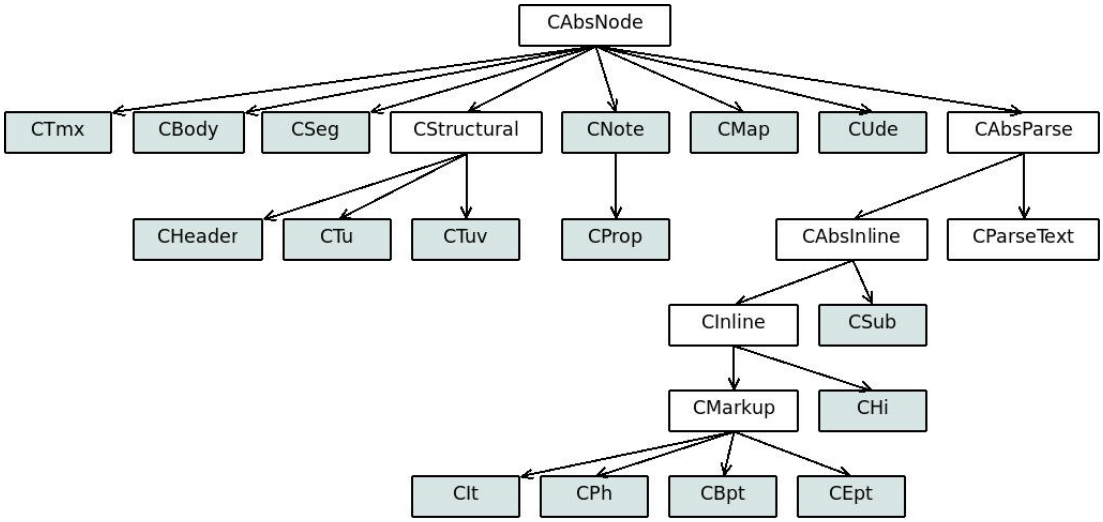
Třída **`CTranslationProject`** definuje rozhraní pro práci s překladovým projektem. Jsou to zejména metody `Translate`, `AutoTranslate`, `GetTranslation` a `AddTranslation`.

Třída **`CAlignmentProject`** definuje metody, které zajišťují alignment (`Realign` a `NewRealign`) a jeho úpravu (`MergeSourceSegments` a `MergeTargetSegments`).

4.2.2 Překladová paměť (třída CMemoryFile)

Třída `CMemoryFile` je použita pro reprezentaci překladové paměti ve formátu TMX a také dat projektů. Uložení projektu do formátu TMX je tedy přirozené a nevyžaduje žádnou konverzi. Třída obsahuje ukazatel na samotnou vnitřní strukturu TMX a soubor metody pro práci s ní.

Vnitřní struktura je poněkud složitější a pro lepší orientaci v ní poslouží obrázek:



Obr. 2: Dědičnost ve vnitřní struktuře překladové paměti

Formát TMX byl z hlediska elementů popsán v kapitole 3.2.2, nyní se na něj podíváme z hlediska implementace. Každý element v TMX stromě má svou reprezentaci v podobě třídy (na obrázku modře), která uchovává jeho data a provádí nad nimi potřebné operace. Všechny třídy elementů dále obsahují odkaz na svého souseda ve stromové struktuře TMX. Třída `CTmx` navíc obsahuje ukazatel na std::vector všech zdrojových segmentů (resp. ukazatelů na ně), nad kterým se pak vyhledává. Abstraktní třída pro všechny elementy je `CAbsNode`.

Třída `CStructural` seskupuje elementy, které uchovávají informace o svém vytvoření, úpravě a typu. `CAbsParse` je abstraktní třída pro veškerý text a formátování, na které můžeme narazit při parsování. `CAbsInline` je abstraktní třída pro vložené elementy, které reprezentují formátování. `CParseText` je pak čistý text, o kterém máme uloženy nějaké informace o jeho obsahu (např. že je to interpunkce). Všechny vložené elementy kromě zvláštního `CSub` mají nadřazenu abstraktní třídu `CInline`. Atributy, které obalují formátovací značky původního dokumentu, jsou potomky třídy `CMarkup`.

4.2.3 Nastavení (třída CSettings)

Třída `CSettings` obsahuje všechna nastavení programu, projektu a přidružené překladové paměti a metody potřebné pro jejich načtení ze souboru a překladové paměti. Tato nastavení jsou:

- zdrojový jazyk;

- cílový jazyk;
- kódování vstupních souborů;
- typ dat, které dokument obsahuje;
- jazyk, v kterém je psán obsah elementů <note> a <prop>;
- typ segmentace;
- výchozí oddělovač vět a odstavců pro případ, kdy již známe správnou segmentaci vět v souborech a chceme provést pouze alignment;
- jméno projektu;
- cesta k projektu;
- cesta ke zdrojovému a cílovému souboru;
- cesta k souboru SRX;
- cesta k souboru s překladovou pamětí;
- cesty k modulům;
- konec řádky, který chceme použít při vytváření a ukládání souborů;
- nastavení pro fuzzy vyhledávání;
- nastavení pro alignment (hodnoty c a s^2).

4.2.4 Rozhraní pro moduly (třída CWrapper)

Třída `CWrapper` načítá moduly a poskytuje funkce, které umožňují volat exportované funkce připojených modulů. Pro úspěšné připojení musí moduly exportovat následující funkce:

- modul vzdálenost funkci `ComputeDistance`;
- modul parser funkce `ParseFile` a `FindMarkup`;
- modul segmenter funkci `SegmentData`;
- modul aligner funkci `AlignData` a volitelně funkce `AlignFiles` a `SimpleAlign`;
- modul exporter funkci `ExportMemory`;
- modul automatického překladu funkci `Translate`.

Podrobněji jsou moduly a funkce popsány v kapitole 4.3.

4.3 Moduly

Moduly jsou uzavřené části aplikace, které se zabývají konkrétním problémem. Zde jsou realizovány pomocí dynamicky linkovaných knihoven. Knihovny musí být napsány v C++ a podle svého typu musí exportovat níže popsané globální funkce. Protože kompilátor C++ obaluje symboly identifikátorů vlastními značkami, je nutné exportované funkce obalit od bloku `extern "C" { … }`.

4.3.1 Parser

Modul parseru pro daný formát představuje schopnost aplikace daný formát otevřít a zpracovat. Aplikace prozatím obsahuje pouze moduly ke zpracování prostého textu a souborů .tra (formáty, které potřebuji nejčastěji), ale díky použití dynamických knihoven není problém přidat další.

Moduly, které obsahují parser musí exportovat globální funkce `ParseFile` a `FindMarkup`.

Funkce `ParseFile` obdrží string s obsahem souboru (v případě formátů, které nejsou přímo textové, je nutné v nastavení přiřadit položce `datatype` hodnotu „`other`“, string pak bude obsahovat cestu k souboru); referenci na STL vektor segmentů, do kterého přidá nalezené segmenty; ukazatel na funkci segmenteru a ukazatel na globální nastavení.

Funkce `FindMarkup` má za úkol odstranit formátovací značky v textu. Na vstupu funkce obdrží string, ve kterém mohou být značky a ukazatel na spojový seznam formátovacích značek, do kterého nalezené značky přidá. Funkce vrací řetězec bez formátovacích značek.

4.3.2 Segmenter

Úkolem segmenteru je rozdělit dodaný text na segmenty vhodné k překladu. Segmenter musí exportovat funkci `SegmentData`. Tato funkce dostane na vstupu text k segmentaci, referenci na STL vektor stringů, do kterého uloží jednotlivé segmenty, a soubor s nastavením. Soubor může být jakéhokoli formátu (záleží na konkrétním segmenteru), ale předpokládá se SRX.

4.3.3 Aligner

Aligner má za úkol přiřadit segmentu ve zdrojovém jazyce segment v jazyce cílovém. Modul obsahující aligner musí exportovat funkci `AlignData`, která dostane na vstupu reference na STL vektory segmentů zdrojového a cílového textu, ukazatel na překladovou paměť, do které převede vzniklé překladové jednotky, a ukazatel na globální nastavení. Dále může (ale nemusí) exportovat funkce `AlignFiles` a `SimpleAlign`. Funkce `AlignFiles` umožní použít aligner přímo na dva soubory místo na vektory segmentů. Funkce `SimpleAlign` je určena k provedení jednoduššího alignmentu v případě, že hlavní funkce z nějakého důvodu selže. Má stejné parametry jako funkce `AlignData`.

Implementace tohoto modulu byla jedním z hlavních cílů této práce, protože přináší funkčnost, která ve většině dostupných aplikací chybí a je to zároveň modul, který nejvíce využiji. Implementovaný aligner používá algoritmus popsaný v kapitole 3.3.2.

4.3.4 Vzdálenost

Modul obsahující funkci pro výpočet vzdálenosti dvou řetězců. Výpočet vzdálenosti řetězců je v případě překladu s pamětí nejpoužívanější funkcí, proto by její výpočet měl trvat pokud možno co nejkratší dobu. Tento modul umožňuje výměnu výchozí funkce, která počítá Levenshtainovu vzdálenost popsanou v kapitole 3.3.1. Modul musí exportovat funkci `ComputeDistance`, která dostane na vstupu dva řetězce a vrátí jejich vzdálenost.

4.3.5 Exporter

Modul umožňující případný export paměti z formátu TMX do formátu implementovaného exporterem. Modul musí exportovat funkci `ExportMemory`, která dostane ukazatel na paměť k exportu.

4.3.6 Automatický překlad

Pokud není v paměti žádný dostatečně podobný segment, může uživatel požádat o pokus o automatické přeložení nějakým systémem automatického překladu. Moduly zprostředkovávající tento překlad musí exportovat funkci `Translate`, která

dostane na vstupu řetězec se zdrojovým textem a dva řetězce udávající z jakého a do jakého jazyka chceme text přeložit. Funkce musí vracet přeložený řetězec.

4.4 Uživatelské rozhraní

Grafické uživatelské rozhraní zatím nebylo implementováno, ale k dispozici je rozhraní v prostředí příkazové řádky. Program se spouští zadáním

```
tras [-volby] vstupní_soubor [další_soubor] [-nastavení]
```

Podrobný popis všech voleb a nastavení naleznete v souboru tras.readme na přiloženém CD.

Ve spuštěném programu jsou následující možnosti práce:

Volba	Akce
n	Přesun na další segment
p	Přesun na předchozí segment
:s	Uložení projektu (a paměti, pokud existuje)
:spa {file}	Uložit projekt jako {soubor}
:d	Smazat aktivní segment
:ns	Přidat nový segment před aktivní segment
:cs	Změnit zdrojový segment
:ct	Změnit cílový segment
:mp	Sloučit segment s předchozím
:mn	Sloučit segment s následujícím
:f {number}	Přesunout se na segment {číslo}
:a	Zobrazit všechny segmenty
:l {number}	Zobrazit {počet} segmentů počínaje aktuálním zobrazeným
:h	Nápověda
:q	Konec

V překladovém projektu:

o	Otevřít aktivní segment
:t	Překládat do první „fuzzy“ shody
:sma {file}	Uložit paměť jako {soubor}
:e {file}	Exportovat cílové segmenty do {soubor}
:em	Export paměti pomocí externí exportní funkce (pokud existuje)

V alignment projektu:

:r	Znovu provést alignment
:msn	Sloučit zdrojový segment s následujícím
:msp	Sloučit zdrojový segment s předchozím
:mtn	Sloučit cílový segment s následujícím
:mtp	Sloučit cílový segment s předchozím
:em	Export paměti pomocí externí exportní funkce (pokud existuje)

Tab. 2: Možnosti uživatelského rozhraní

4.5 Rozdělení zdrojového kódu

Zdrojový kód je rozdělen do souborů s přihlédnutím k jeho funkci. Moduly mají každý svůj vlastní podadresář, ale jsou součástí vlastního kódu, aby bylo možné je přilinkovat i přímo do aplikace. Přehled jednotlivých souborů:

`CProject.h` – projekty

`CMemoryFile.h` – překladová paměť

`CTMXStructural.h` – třída `CAbsNode` a všechny třídy strukturálních elementů

`CTMXInline.h` – třídy `CAbsParse`, `CParseText` a `CAbsInline` a všichni její potomci

`CTMXAttribute.h` – třída `CAttribute` reprezentující atributy TMX elementů

`CTMXParser.h` – třída `CTMXParser`, která zajišťuje správné načtení souborů TMX

`CSettings.h` – nastavení

`CWrapper.h` – rozhraní pro moduly

`CMyException.h` – výjimky

`lang.h` – definice řetězcových konstant

`functions.h` – pomocné funkce

`main.cpp` – inicializace a rozhraní příkazové řádky

`distance\distance.h` – modul vzdálenosti

`alignment\align.h` – modul aligner

`parsers\tra.h` – modul parser formátu .tra

`segmenters\simple.h` – modul jednoduchého segmenteru

4.6 Kompilace

Pro systém Windows probíhala kompilace a linkování automaticky za použití vývojového prostředí Code::Blocks[19] a kompilátoru G++ a zejména z důvodu ladění jsem linkovala moduly přímo do výsledné aplikace. Výsledkem je tak jeden

spustitelný soubor, který podporuje pouze moduly, které byly v době kompilace součástí kódu. Vývojové prostředí Code::Blocks ale podporuje více cílů kompilace a tak je rozdělení do modulů pouze otázkou drobné úpravy konfigurace projektu.

V Linuxu byla situace o něco složitější, protože Code::Blocks nepodporuje generování makefile. Kompilace a linkování je zde řízeno ručně psaným makefile skriptem, který aplikaci dělí na jednotlivé moduly.

Kapitola 5
Závěr

Mým cílem bylo vytvoření systému s překladovou pamětí a alignmentem vět, který by splňoval požadavky uvedené v kapitole 2.4. Tento cíl jsem do jisté míry splnila vytvořením aplikace TRAS, která je součástí této práce a najdete ji na přiloženém CD.

V současnosti je aplikace funkční pod operačním systémem Linux, s omezením i pod systémem Windows. Na platformě Windows plynou omezení z příkazové řádky systému, na které se mi nepodařilo vyřešit problémy s kódováním. V obou systémech je však schopna načítat moduly a rozšiřovat tak své funkce. Celkově je aplikace pro mé účely použitelná, ale zasloužila by si rozšířit, především o grafické uživatelské rozhraní.

5.1 Srovnání s existujícími aplikacemi

Nakonec si dovolím malé srovnání s aplikacemi uvedenými v kapitole 3.1:

	Otevření paměti v TMX	Vytvoření TMX	Multi-platformní	Alignment	Fuzzy vyhledávání	Neváza-nost na aplikace	GUI
Anaphraseus	import	export	☑	☒	☑	☒	OO
OmegaT+	☑	☑	☑	☒	☑	☒	☑
OLT	import	export	☑	další program	☑	☑	☑
Transolution	☒	☒	☑	☒	☑	☑	☑
Translate Toolkit	☒	export	☑	☒	☑	☑	☑
Transit	?	?	☑	☒	☑	☑	☑
TRAS	☑	☑	☑	☑	☑	☑	☒

Tab. 3: Srovnání s existujícími aplikacemi

Z tabulky 3 vyplývá, že ačkoli systém TRAS nemá prozatím grafické uživatelské rozhraní, má i přesto svým uživatelům co nabídnout.

5.2 Další vývoj

Díky svému návrhu má aplikace široké možnosti rozšíření, zejména přidáním dalších modulů. V blízké budoucnosti předpokládám rozšíření o grafické uživatelské rozhraní. Z modulů pak připojení některého už existujícího silnějšího nástroje na segmentaci, připojení k volně dostupnému systému automatického překladu – např. Google Translate a rozšíření nabídky podporovaných souborů alespoň o formát HTML.

V delším časovém horizontu pak implementaci konvertoru pro XLIFF, podporu obecného XML a další.

Literatura

[1] OASIS XML Localisation Interchange File Format (XLIFF) [online] (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xliff)

[2] Translation Memory eXchange (TMX) [online] (http://www.lisa.org/Translation-Memory-e.34.0.html)

[3] Segmentation Rules eXchange (SRX) [online] (http://www.lisa.org/Segmentation-Rules-e.40.0.html)

[4] Anaphraseus [online] (http://anaphraseus.sourceforge.net/)

[5] OmegaT+ [online] (http://omegatplus.sourceforge.net/)

[6] Open Language Tools [online] (https://open-language-tools.dev.java.net/)

[7] Transolution [online] (http://transolution.python-hosting.com/)

[8] The Translate Toolkit [online] (http://translate.sourceforge.net/wiki/toolkit/index)

[9] Transit [online] (http://www.star-group.net/star-www/description/transit/star-group/csy/star.html)

[10] OSCAR, LISA's Standards Committee [online] (http://www.lisa.org/OSCAR-LISA-s-Standa.79.0.html)

[11] TMX 1.4b Specification [online] (http://www.lisa.org/fileadmin/standards/tmx1.4/tmx.htm)

[12] Hall P. A. V., Dowling G. R., „Approximate string matching", *Computing Surveys*, 12(4), 381–402 (1980).

[13] Brown P., Lai J., Mercer R., „Aligning Sentences in Parallel Corpora", in *Proceedings of ACL-91*, 1991.

[14] Kay M., Röscheisen M., „Text-Translation Alignment", in *Computational Linguistics*, Vol. 19, No. 1, 1993.

[15] Gale W., Church K., „A Program for Aligning Sentences in Bilingual Corpora“, in *Proceedings of ACL-91*, 1991

[16] Knihovny GTK+ [online]
(http://www.gtk.org/)

[17] Knihovny wxWidgets [online]
(http://www.wxwidgets.org/)

[18] Rozhraní knihoven GTK+ pro C++ [online]
(http://www.gtkmm.org/)

[19] Vývojové prostředí Code::Blocks [online]
(http://www.codeblocks.org/)

Příloha A

Ukázka dokumentu ve formátu TMX

```
<?xml version="1.0"?>
<!-- Example of TMX document -->
<tmx version="1.4">
    <header
      creationtool="XYZTool"
      creationtoolversion="1.01-023"
      datatype="PlainText"
      segtype="sentence"
      adminlang="en-us"
      srclang="EN"
      o-tmf="ABCTransMem"
      creationdate="20020101T163812Z"
      creationid="ThomasJ"
      changedate="20020413T023401Z"
      changeid="Amity"
      o-encoding="iso-8859-1"
    >
        <note>This is a note at document level.</note>
        <prop type="RTFPreamble">{\rtf1\ansi\tag etc...{\fonttbl}</prop>
        <ude name="MacRoman" base="Macintosh">
            <map unicode="#xF8FF" code="#xF0" ent="Apple_logo" subst="[Apple]"/>
        </ude>
    </header>
    <body>
        <tu
          tuid="0001"
          datatype="Text"
          usagecount="2"
          lastusagedate="19970314T023401Z"
        >
            <note>Text of a note at the TU level.</note>
            <prop type="x-Domain">Computing</prop>
            <prop type="x-Project">P&#x00E6;gasus</prop>
            <tuv
              xml:lang="EN"
              creationdate="19970212T153400Z"
              creationid="BobW"
            >
                <seg>data (with a non-standard character: &#xF8FF;).</seg>
            </tuv>
            <tuv
              xml:lang="FR-CA"
              creationdate="19970309T021145Z"
              creationid="BobW"
              changedate="19970314T023401Z"
              changeid="ManonD"
            >
                <prop type="Origin">MT</prop>
                <seg>donn&#xE9;es (avec un caract&#xE8;re non standard: &#xF8FF;).</seg>
            </tuv>
        </tu>
        <tu
```

```
            tuid="0002"
            srclang="*all*"
          >
              <prop type="Domain">Cooking</prop>
              <tuv xml:lang="EN">
                  <seg>menu</seg>
              </tuv>
              <tuv xml:lang="FR-CA">
                  <seg>menu</seg>
              </tuv>
              <tuv xml:lang="FR-FR">
                  <seg>menu</seg>
              </tuv>
          </tu>
      </body>
</tmx>
```

Příloha B

Obsah přiloženého CD

Součástí této práce je přiložené CD s následující obsahem:

- Soubor **bc.pdf** je elektronická verze tohoto dokumentu.
- V souboru **read.me** jsou popsány pokyny k instalaci.
- Soubor **tmx14.dtd** obsahuje DTD formátu TMX ve verzi 1.4.
- V adresáři **Documentation** se nachází programátorská dokumentace vygenerovaná systémem Doxygen (www.doxygen.org).
- V adresáři **Windows** naleznete vše potřebné k provozování aplikace pod operačním systémem Windows.
- Adresář **Code** obsahuje zdrojové kódy aplikace.